(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 March 2001 (29.03.2001)

PCT

(10) International Publication Number
WO 01/22682 A2

(51) International Patent Classification[7]: H04L 29/00

(21) International Application Number: PCT/US00/26084

(22) International Filing Date:
22 September 2000 (22.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/155,354	22 September 1999 (22.09.1999)	US
09/668,498	22 September 2000 (22.09.2000)	US

(71) Applicant: STREAMING21, INC. [US/US]; Suite 1, 170 Knowles Drive, Los Gatos, CA 95032 (US).

(72) Inventors: NGAI, Ray, T., F.; 3448 Bonita Avenue, Santa Clara, CA 95051 (US). LEE, Horng-Juing; 6452 Trinidad Drive, San Jose, CA 95120 (US). LEE, Yen-Jen; 3643 Madison Common, Fremont, CA 94538 (US). LIN, Joe, M-J.; 2045 Paseo del Sol, San Jose, CA 95124 (US).

(74) Agents: LEBLANC, Stephen, J. et al.; Majestic, Parsons, Siebert & Hsue P.C., Suite 1100, Four Embarcadero Center, San Francisco, CA 94111-4106 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/22682 A2

(54) Title: METHOD AND SYSTEM FOR PROVIDING REAL-TIME STREAMING SERVICES

(57) Abstract: A method and system for streaming media delivery is disclosed. The invention allows for providing streaming media and interfacing with generic file systems, database, and other components.

METHOD AND SYSTEM FOR PROVIDING REAL-TIME STREAMING SERVICES

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority from provisional patent application 60/155,354 entitled METHOD AND SYSTEM FOR PROVIDING REAL-TIME STREAMING VIDEO SERVICES filed 09/22/99.

The above referenced application(s) are incorporated herein by reference for all purposes. These prior applications, in some parts, may indicate earlier efforts at describing the invention or describing specific embodiments and examples. The present invention is therefore best understood as described herein.

FIELD OF THE INVENTION

The invention generally relates to methods and/or devices related to data transmission. More particularly, the invention in various aspects, relates to methods and systems for providing real-time multimedia streaming over a communication network.

BACKGROUND OF THE INVENTION

The Internet is a rapidly growing communication network of interconnected computers around the world. Together, these millions of connected computers form a vast repository of hypermedia information that is readily accessible by users through any of the connected computers from anywhere and anytime. As there is an increasing number of users who are connected to the Internet and surf for various information, there meanwhile is created tremendous demand for more content to be available and methods to deliver that content on the Internet. Currently, the most commonly available media information that is deliverable on the Internet may include text information, images and graphics, videos and audio clips.

Typically, continuous or streaming media information, such as videos and audio, comes in the form of streaming data. For example, streaming video is a sequence of "moving images" that are sent in compressed form over a network and displayed by a viewer as they arrive. With streaming video or streaming media, a Web user does not

have to wait to download a large file before seeing the video or hearing the sound. Instead, the media is sent in a continuous stream and is played as it arrives, though generally with some delay for buffering the media. The user uses a player (either hardware or software) to play the media. In the case of a software player, a special program is executed to uncompress and send video data to a display screen and audio data to speakers.

There are many purposes of providing streaming media information on demand over a data network including a cable network or the Internet. Consumers may enjoy movies or receive latest news at their leisure time rather than at fixed schedule. Businesses implement streaming media information on their Intranet to accelerate high-value processes by providing faster and better quality communication with employees, business partners, and customers. They gain the flexibility to take internally created or externally acquired video information (content) and make it available to both internal and external customers. In so doing, information can be shared among distribution partners and suppliers and can even be repackaged for new business alliances and customers.

Users are constantly demanding access to streaming media information on a truly unprecedented scale, which can generate enough audio/video demands to completely overwhelm a video server that provides the media information as well as an unprotected data network. As a result, the quality of services of delivering the media information suffers and the users are frustrated. Thus there have been high demands for video server systems that can not only handle hundreds or thousands of requests at any given time but also guarantee the quality of services of delivering the media information.

Streaming services generally are provided via Web Servers or via specialized Streaming Media Servers. (See, for example, *www.microsoft.com/ntserver/mediaserv/exec/comparison/WebServVStreamServ.asp.*) Specialized streaming servers typically include streaming media delivery software for managing the real-time deliver of data and also include a specialized file system or format for storing data prior to streaming. Typically, the link between a particular specialized file system and the logic functions that managing the streaming sessions is very tight and is highly optimized.

SUMMARY

The present invention, in specific embodiments, involves an innovative software engine (at times referred to herein as Real-Time Streaming Engine (RTSE)) that handles large-scale real-time streaming services. According to specific embodiments of the present invention, each stream can be understood as a user access from a client on one computer to a server on the other computer. Users access real-time data such as audio or video from a server and the server delivers, or streams, the real-time audio, video or other data with certain time constraints to client via a computer or communications network.

Unlike prior streaming engines, an RTSE according to the present invention has an architecture that allows the central engine software to work with a variety of streaming file systems and conventional file systems as well as a variety of live-data hardware and/or software encoders. The present invention accomplishes this through a innovative combination of known and innovative components into a new architecture for providing streaming services. In various aspects, the invention includes innovative methods for providing and/or managing streaming media according to this flexible design.

The invention therefore in specific aspects handles streaming video/audio signals that can be played on various types of video-capable terminal devices operating under any type of operating system and regardless of what type of players are preinstalled in the terminal devices.

According to specific embodiments of the present invention, the RTSE accomplishes effective handling of streaming sessions with a flexible and extensible architecture by dividing tasks among a plurality of managers, such as Reception Manager, Streaming File Manager, Encoding File Manager, Network Manager, Administration Manager, Streaming Manager, Database Manager, or User Interaction Manager. While not all implementations of servers according to the invention will include all of the managers, managers present in a server can be understood to collaborate with each other to achieve large-scale streaming service.

Using multiple managers in an architecture according to the invention, the present invention has the ability to handle various types of file systems (e.g. the software according to the invention can be adapted to different streaming and

conventional file systems, thus the architecture is able to be adapted to a generic streaming file system) and various types of software and hardware encoding modules, database engines, networking protocols, real-time media, streaming engines, etc.

A server device, when loaded with and executing the server module, will provide large scale real-time streaming media services (which can include both delivering and/or accepting streaming data) to support a large number of streaming sessions without compromising the quality of services in delivering the streaming media information over a network. The network may include a cable network, a local area network, a network of other private networks and the Internet.

The invention as described further below can be implemented in numerous ways including a method, a computer readable medium, a system, and/or an apparatus. The advantages of the invention are numerous and certain embodiments of the invention can have one or more of the following advantages. One advantage of the invention is that it provides a great deal of flexibility over the quality of service provided with respect to delivering real-time streaming media data over a network. Another advantage of the invention is that all that task-specific managers collaborate with each other to achieve the large-scale real-time streaming service. Still another advantage of the invention is that implementation and configuration of all the managers are easy to manage and install. Although the media delivery system is described herein based on video streaming signals, those skilled in the art can appreciate that the description can be equally applied to continuous data delivery that may include streaming audio.

The detailed description of the present invention includes numerous specific details that are set forth in order to provide a thorough understanding of the present invention. However, it will become obvious to those skilled in the art that the present invention may be practiced without these specific details. In other instances, well known methods, procedures, components, and circuitry have not been described in detail to avoid unnecessarily obscuring aspects of the present invention.

The present invention is presented largely in terms of procedures, steps, logic blocks, processing, and other symbolic representations that resemble data processing devices. These process descriptions and representations are the means used by those experienced or skilled in the art to most effectively convey the substance of their work to others skilled in the art. The method along with the system to be described in detail

below and the appendix is a self-consistent sequence of processes or steps leading to a desired result. These steps or processes are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities may take the form of electrical signals capable of being stored, transferred, combined, compared, displayed and otherwise manipulated in a computer system or electronic computing devices. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, operations, messages, terms, numbers, or the like. It should be borne in mind that all of these similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following description, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "verifying" or "displaying" or the like, refer to the actions and processes of a computing device that manipulates and transforms data represented as physical quantities within the device's registers and memories into analog output signals via resident transducers.

It is well known in the art that logic or digital systems and/or methods can include a wide variety of different components and different functions in a modular fashion. The following will be apparent to those of skill in the art from the teachings provided herein. Different embodiments of the present invention can include different combinations of elements and/or functions. Different embodiments of the present invention can include actions or steps performed in a different order than described in any specific example herein. Different embodiments of the present invention can include groupings of parts or components into larger parts or components different than described in any specific example herein. For purposes of clarity, the invention is described in terms of systems that include many different innovative components and innovative combinations of innovative components and known components. No inference should be taken to limit the invention to combinations containing all of the innovative components listed in any illustrative embodiment in this specification. The functional aspects of the invention, as will be understood from the teachings herein, may be implemented or accomplished using any appropriate implementation environment or programming language, such as C++, Cobol, Pascal, Java, Java-script, etc. All

publications, patents, and patent applications cited herein are hereby incorporated by reference in their entirety for all purposes.

The invention and various specific aspects and embodiments will be better understood with reference to the following drawings and detailed descriptions. In different figures, similarly numbered items are intended to represent similar functions within the scope of the teachings provided herein. In some of the drawings and detailed descriptions below, the present invention is described in terms of the important independent embodiment of delivering visual and/or audio content. This should not be taken to limit the invention, which, using the teachings provided herein, can be applied to other streaming data content. For purposes of clarity, this discussion refers to devices, methods, and concepts in terms of specific examples. However, the invention and aspects thereof may have applications to a variety of types of devices and systems. It is therefore intended that the invention not be limited except as provided by the attached claims and equivalents.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates an exemplary configuration of a data network in which the present invention may be practiced.

FIG. 2 is a general diagram showing various components of a system according to various specific embodiments of the present invention.

FIG. 3 is a block diagram showing a general architecture arrangement of a streaming service according to specific embodiments of the present invention.

FIG. 4 is a block diagram illustrating a further example architecture of a streaming service according to specific embodiments of the invention.

FIG. 5 is a block diagram illustrating a further example architecture of a streaming server according to specific embodiments of the invention.

FIG. 6 is a block diagram illustrating an example system diagram of a Streaming File Manager according to specific embodiments of the invention.

FIG. 7 is a block diagram illustrating an example encoding manager according to specific embodiments of the invention.

FIG. 8 is a flow chart illustrating example encoder operation according to specific embodiments of the invention.

FIG. 9 is a block diagram showing a representative example logic device in which aspects of the present invention may be embodied.

DESCRIPTION OF SPECIFIC EMBODIMENTS

General Description of Data Network

Referring to the drawings, FIG. 1 illustrates an exemplary configuration in which the present invention may be practiced. Central video server **102** together with a video database **104** is a video source comprising video files that can be accessed on demand. A video source can also be provided by live encoders as described below. As used herein, video files or titles are referred to any video footage, video films and/or video/audio clips that typically in a compressed format such as MPEG or MP3. It should be noted, however, that the exact format of the video files do not affect the operations of the present invention. As will be noted and appreciated, the present invention applies to any formats of the video files.

Preferably data network **106** is a data network backbone, namely a larger transmission line. At the local level, a backbone is a line or set of lines that local area networks connect to for a wide area network connection or within a local area network to span distances efficiently (for example, between buildings). On the Internet or other wide area network, a backbone is a set of paths that local or regional networks connect to for long-distance interconnection. Coupled to data network A **106**, there are two representative proxy servers **108** and **110** that each service representative terminal devices **116-119** via data network **112** and **114** respectively. It should be noted video server **102** named "central" does not necessary mean that video server **102** is only served as a repository of all the video titles. In some case, central video server **102** may service terminal devices directly and may retrieve other video titles from other servers such as proxy servers **108** and **110**. A compiled version and linked version of the present invention, as a server version, may be employed or installed in any of servers **102, 108** and **110**.

Data network **112** and **114** are typically the Internet, a local area network or phone/cable network through which terminal devices can receive video files. The terminal devices may include, but not be limited to, multimedia computers (e.g. **116** and **119**), networked television sets or other video/audio players (e.g. **117** and **118**). Typically the terminal devices are equipped with applications or capabilities to execute

and display received video files. For example, one of the popular applications is an MPEG player provided in WINDOWS 98 from Microsoft. When an MPEG video file is received in streaming from one of the proxy servers, by executing the MPEG player in a multimedia computer, the video file can be displayed on a display screen of the computer.

To receive a desired video, one of the terminal devices may send in a request that may comprise the title of the desired video or another identifier. Additionally the request may include a subscriber identification if the video services allow only authorized access. Upon receiving the request, the proxy server will first check in its cache if the selected video is provided therein, meanwhile the request is recorded by a network manager in the server module. The description and representation provided herein are the common means used by those experienced or skilled in the art to most effectively convey the substance of their work to others skilled in the art. In other instances, well known methods, procedures, components, and circuitry have not been described in detail to avoid unnecessarily obscuring aspects of the present invention.

As a result of the processes carried by the server module, the selected video will be provided as a streaming video to the terminal device if the entire video is in the cache. Otherwise the proxy server proceeds to send a request to video central server **102** for the rest of the video if there are some units of the video in a cache memory of the proxy server or the entire video if there is no any unit of the video.

FIG. 2 illustrates various components that may be included in a server corresponding to server **102, 108** or **110** in FIG. 1. According to one embodiment of the present invention, the server is loaded with a server module implemented in a compiled and linked version of an embodiment of the present invention, when the server module is executed by the processor, the server will perform the desired features as described above and further in the description below.

Design Concept

FIG. 3 is a block diagram showing a general architecture arrangement of a streaming service according to specific embodiments of the present invention. This general architecture illustrates in concept how the present invention achieves the flexible streaming server design by providing a central (or core) Reception Manager that provides an interface for other parts of the streaming server. The architectural design

according to the present invention can be further understood by considering the modules/managers described below.

FIG. 4 is a block diagram illustrating a further example architecture of a streaming service according to specific embodiments of the invention. This figure shows an arrangement of a server in a specific embodiment.

FIG. 5 is a block diagram illustrating a further example architecture of a streaming server according to specific embodiments of the invention. This figure shows an arrangement of a server in a specific embodiment providing additional details and components.

Core Reception Manager (RM)

RM defines a set of communication principles (protocols) for other managers to use in collaborating with each other to complete streaming/management requests from clients/consoles (e.g. users or administrators). This is the core manager in the RTSE architecture to control/integrate loosely coupled managers to work together seamlessly and coherently. RM provides an RTSE an extension capability because the RM design allows easier adopting/integrating of streaming, file, encoding, network, database, administration and/or user-interaction managers, whether additional proprietary managers or written by third parties.

In a further embodiment, the RM provides the user authentication and access control to provide different level of security with interface to various formats of user profile repositories. Regarding RTSE messaging, the RM is the middleman to gather, handle, store, and/or redistribute messages among managers. Different types of managers can communicate and collaborate with each other with predefined disciplines to achieve real-time tasks.

In various specific embodiments, tasks of the RM include one or more of the followings: (1) retrieving stored data from real-time Streaming File Manager (FM); (2) retrieving live data from real-time Encoding File Manager (EM); (3) processing requests from real-time Network Manager (NM) via a network; (4) processing requests from real-time Administration Manager (AM) via a network; (5) processing database requests to/from Real-time Database Manager (DBM); (6) processing user interaction requests from User Interaction Manager (UIM); (7) processing requests from Real-time

Streaming Manager (SM); (8) processing error handling of the whole engine; and (9) processing special events of the whole engine.

File Managers (FM)

FMs manage stored real-time data in storage systems such as Disk Array, CD-ROM Jukebox, MO Jukebox, and tape storage. FM schedules concurrent stream requests from Streaming Manager and is designed to satisfy all *accepted* stream requests in real-time fashion.

FM includes several components such as Admission Control module, Real-time scheduler, Media Disk module, and Cache module. FM communicates with subsystems such as a proprietary Streaming File System (SFS), Window File System (WFS), and/or other 3rd party file systems.

In a particular embodiment, FM provides a generic Applications Programming Interface (API) to handle various underlying file systems. As understood in the art, an API is a set of functions, possibly with associated data (such as class definitions in C++), that programs or other logic modules can use to access system provided services, such as operating system services, file system services, etc.

Streaming Manager (SM)

This is the streaming delivery module of RTSE. SM handles requests from Network Manager and performs appropriate actions to call Streaming File Manager and Encoding File Manager as needed. Streaming Manager receives data from either Streaming File Manager or Encoding File Manager and schedules data delivery to remote clients via any pre-negotiated networking protocols.

Encoding File Manager (EM)

EM and EM instances bridge different vendors of hardware and software encoders to Reception Manager to handle real-time encoding of audio and video and deliver the real-time data to Reception Manager using RTSE's generic File System API. EM schedules concurrent stream requests from Streaming Manager. EM is required to satisfy all accepted stream requests in real-time fashion. The Encoding File Manager includes several components as discussed in more detail below.

Network Manager (NM)

NM listens to incoming requests from remote clients and performs the corresponding actions for each request. NM handles single listen thread or multiple

collaborating threads to work together to improve the load balance and performance of the RTSE.

Data Manager (DBM)

DBM handles requests related to user management profile, log and event profile, file system profile, and media meta-files. DBM isolates database-related requests with a generic API to access information stored on data servers such as (but not excluding) Jet Database, SQL Database server, or other database engines or in plain text file data storage.

Administration Manager (AM)

AM listens to incoming monitoring or management requests from remote management consoles and handles the corresponding management jobs according to the incoming requests.

User Interface Manager (UIM)

UIM queries Reception Manager for management, monitoring, status report, and more and is designed to provide a user-friendly graphical interface to expose the functionality of the whole RTSE through the Reception Manager. UIM enables a verified user (such as a network administrator) to perform user authentication, server control, media object management, user profile management & server settings configuration.

EXAMPLE SPECIFIC EMBODIMENT

FIG. 4 is a block diagram illustrating a further example architecture of a streaming service according to specific embodiments of the invention. As further elaborated in FIG. 4, a Real-time Reception Manager according to specific embodiments of the present invention, may be wrapped by a set of core APIs.

FIG. 4, as will be understood from the teachings provided herein, shows additional details and structure of an example server module constructed according to the architectural principals illustrated in FIG. 3, with the general API wrapping of FIG. 3 further specified as the four different APIs illustrated. As will be understood in the art, the distinctions of the different APIs shown in FIG. 4 can be understood as an implementation feature of specific systems, and API definitions, in some embodiments, may be universal and accessible to various external managers.

FIG. 5 is a block diagram illustrating a further example architecture of a streaming server according to specific embodiments of the invention. This figure shows an as implemented configuration of a server according to the present invention, and provides further details of data flow.

RECEPTION MANAGER (RM)

As described above, RM defines a set of communication principles (protocols) for other managers to use to complete streaming/management requests.

In particular embodiments, RM does this by defining various standard interfaces (e.g. APIs) for use by each of the interacting managers and then by providing the necessary logic to translate a access request coming from, for example, the network manager through the server reception API, to, for example a File Manager to determine the availability of a file location, and then to a Streaming Manager to handle the time critical exchange of data.

For many kinds of communications, all inter-manager communications requests can be understood to pass through the RM. Thus, for most types of requests, each manager communicates its requests to RM and receives any necessary responses from RM. RM internally decides which managers to invoke to service a particular request and invokes those managers and gathers results to present to the requesting process.

As shown in FIG. 4 and in FIG. 5, an RM according to specific embodiments can be understood to include the following components.

APIs

Server Reception API (SRAPI): provide an interface to communications with users, both client users and administrators. In specific embodiments, this API can be understood to exchange requests and data with one or more Network Managers, such as an HTTP Interface, FTP Interface, or RTSP interface and/or with one or more User Interface/Administration Managers that can provide administrative access either through a local system console or through an network protocol that allows for system administration, such as SNMP.

In specific embodiments, SRAPI provides various interface functionality. General examples of the types of functions that can be defined within SRAPI and called by managers that interact with SRAPI are provided below. Other functions may also be defined within SRAPI. For each function, a manager calls the function with the

arguments defined for the function and receives function-defined returned results. The various managers are responsible for translating between the specific device or protocol that they are managing (such as HTTP, for example) and the SRAPI interface. Each called SRAPI function is received by the RM, which coordinates a response with other managers and through other APIs, as discussed below, and then returns any results back to the calling manager in the API defined format. The manager then translates any results to an appropriate format or device (such as HTTP). SRAPI can include functions such as those shown below. For these and other function examples provided, the function names will indicate to those of skill in the art generally what the functions do. The arguments and/or data structures passed by the functions will vary with different implementations. Coordination and distribution of functional tasks among managers is generally handled by RM.

It will be understood from the teachings herein, that managers for various protocols, encoders, services or file systems, will include logic for translating valid API requests to the command format or protocol required by the underlying entity to which those managers relate and will include logic for translating back from the entities native response format to a format specified by the API. This generally will be a straight forward programming tasks given a defined set of APIs and a particular native format. Examples of functions specified by SRAPI include:

Server Reception Specific

Init(); UnInit()

User Access/Management

UserLogin(); UserLogout(); AddUser(); AddGroup()

Server Service Specific

StartService(); StopService(); GetServiceStatus()

File System (Fs) Specific

GetFileInfo(); CopyFile(); GetFsSession(); GetFsPerformance(); GetDiskInfo()

Streaming System Specific

ConnectStream(); OpenStream(); CloseStream(); PauseStream()

EventLog & SessionLog

AddEventLog(); GetLatestEventLogs()

Server Messaging

MsgReceiverRegister(); MsgReceiverUnRegister()

It will be understood from the teachings provided herein that the arguments and returned values for these functions may be variously defined in specific implementations according to the invention.

Data Management API (DAPI): provides access to one or more various data sources to facilitate various functions, such as, in specific embodiments, user authentication and access, session and event logging, and storage and retrieval of file or streaming clip data. General examples of the types of functions that can be defined within DAPI are provided below. Other functions may also be defined within DAPI. For each function, generally the RM calls the function with the arguments defined for the function and receives function-defined returned results. The various managers are responsible for translating between the specific device or protocol that they are managing (such as flat text storage or a particular DBMS, for example) and the DAPI interface. DAPI can include functions such as:

GetUserInfo(); AddUser(); UpdateUser(); DeleteUser();

LinkUserToGroup(); AddSessionLog(); GetLatestEventLog();

In particular embodiments, a DAPI may also include a set of functions for storing and retrieving data relating to files or clips stored in 3rd party file systems, where that data is not supported by those file systems. This data might include such items as *clip duration, clip ID, owner, # of frames,* etc. For file systems that do not support storing this type of data within the file system itself, a File System Manager will ensure that this additional data is stored and can be retrieved from a separate data source, through functions provided in DAPI.

File Systems API (FSAPI): provides a standard interface to various underlying file systems and/or encoding systems and their associated managers. General examples of the types of functions that can be defined within FSAPI are provided below. Other functions may also be defined within FSAPI. For each function, generally either RM or a streaming manager as described herein calls the function with the arguments defined for the function and receives function-defined returned results. The various File System Managers are responsible for translating between the specific File System that they are

managing and for providing various buffering and caching functions for that file system, as described further herein. FSAPI can include functions such as:

System Management

FsStart(); FsStop()

Disk Setup and Configuration

FsGetDiskInfo(); FsFormatDisk(); FsConfig()

State/Session/Performance

FsGetStatus(); FsGetPerformance(); FsGetSession(); FsKillSession()

Directory/FileInfo

FsGetFileInfo(); FsSetFileUpdatableInfo(); FsCreateDirectory()

File movement

FsCopy(); FsRename(); FsDelete(); FsImport(); FsExport()

File data operation

FsOpen(); FsRead(); FsWrite(); FsClose(); FsEndOfFile(); FsSetPosition()

Streaming Systems API (SSAPI): provides a standard interface to underlying streaming services and their associated managers. General examples of the types of functions that can be defined within SSAPI are provided below. Other functions may also be defined within SSAPI. For each function, generally RM calls the function with the arguments defined for the function and receives function-defined returned results. The various Streaming Systems Managers are responsible for translating between the specific Streaming Service that they are managing and for providing various buffering and caching functions for that file system, as described further herein. SSAPI provides access to managers that handle time-critical exchanges of data. This API provides a set of objects and/or functions that can be accessed by RM and in some cases by FM relating to streaming services. Thus, different streaming service providers can be accessed through a server according to the present invention, while only modifying the translations provided by the managers. SSAPI can include functions such as:

StmStart(); StnStop(); StmGetStatus(); StmGetPerformance(); StmGetSession(); OpenStm(); CloseStm();

Internal Components of RM

RM is shown in FIG. 4 and in FIG. 5 with a number delineated internal functional modules. These are illustrated to provide further understanding of the internal functions handled by the RM, though the RM may in fact be implemented as a single logic routine without the individually delineated modules shown.

The three masters illustrated are can be understood as distributing commands to specific underlying file or streaming managers. These masters handle broadcasting commands and gathering results across groups of specific managers, and also perform functions that require multiple managers to work together concurrently.

Server Master: This module can be understood to provide various accessing and scheduling logic within the RM, including communication between the Streaming System Master and the File System Master. This module is a part of the RM and can be understood as a helper module in the RM.

File System Master: This module can be understood to provide various overall control and interface functions to of various file system devices.

Streaming System Master: This module can be understood to provide various overall control of streaming sessions provided by the RTSE.

User Authentication & Access Control: This logic/module can be understood as the internal RM logic that handles these functions and communicates with other managers. This function also could be understood as a separate manager communicating through the User Profile API to the data storage for the user profile.

Messaging & Logging: This logic/module can be understood as the internal RM logic that handles these functions and communicates with other managers. This function also could be understood as a separate manager communicating through the Event Log API and Session Log API to various possible data storage for this data.

RM General Operation

RM can be understood as performing an overall manager/coordinator role for RTSE plus a reception role for receiving outside requests directed to the RTSE and coordinating the servicing of those requests. In specific embodiments, RM performs message routing (e.g. job distribution), co-ordination, and outside-interfacing duties. RM generally has application and packaging knowledge that other managers do not. RM performs functions such as:

Initialization - bring up the appropriate modules/managers for different applications of the RTSE. After the initialization, all managers are ready to report their current status to Reception Manager for central control.

Access Control - Ensure different level of access control for various Function calls and requests to playback or record various media clips, login/logout, etc., so that other managers do not have to perform these functions.

User Interface control – RM can perform various tasks based on data in the user profile, such as translating between file system paths and a user home directory path. RM can also perform disk quota checking for users and groups; License Key Checking (Expiration, Application Type, etc.); Format Guard (disable this functionality according to License Key); System Guard (disable this functionality according to License Key).

RM Execution Operation

During its execution life time, the RM and its relationship with other managers can be understood as follows. When the Server is initializing/uninitializing, RM determines the application packaging and co-relation of the other managers, because managers can be packaged differently for different specific implementations.

First of all, a main executable (e.g. a server UI) will pass RM the application type to be launched. Then, RM will find the software license key and verify it. Verifications can include such things as application, expiry date, machine name, version, platform, server maximum capability, etc. RM reads some parameters/settings from non-volatile storage (e.g. an OS Registry) Parameters can include data for such things as security level; event logging storage type (DB file, text file, etc.) corresponding to different Event Logging Modules; User Profile type (DB file, LDAP, text file, etc.) corresponding to different User Profile Modules; streaming session logging storage type (DB file, or W3c format text file, etc.).

RM will then initialize the appropriate managers in appropriate sequence. In particular embodiments, RM initializes a Server Master (a helper inside RM), which is responsible to bring up File System Master first, then Streaming System Master. The File System Master (another helper inside RM) will bring up the appropriate/adequate file system managers (e.g. according to license and application) and can also perform inter-file-systems functions (e.g. Xcopy) and group functions. Streaming System

Master (another helper inside RM) will initialize the appropriate/adequate streaming managers (e.g. according to license and application) and can also perform inter-streaming-systems functions and group functions.

The Server Master will pass a handle of the File-System-Master to each of Streaming Managers, so that individual Streaming Managers can talk directly to File-System-Master (therefore, all file managers) to perform data retrieval and recording.

RM will also bring up other necessary modules, such as User Profile (Authentication and Access Control); one or more of the Messaging and Logging modules, and one or more of Session Logging Modules. These modules can communicate with Database Management APIs to store or retrieve data in a variety of data formats.

RM will then run through an initialization check-list to ensure system-integrity, and screen out problems in advance. This check-list can include such tasks as:

test if the operator launching the server could perform low-level disk access;

ask the appropriate File Manager to test the machine's disk capability/availability;

ask the appropriate Streaming Manager to test the machine's network capability/availability;

ask the appropriate User-Profile Manager, and challenge it if it does not contain adequate tables/accounts/groups.

When terminating the server, unininitializing is also taken care of by RM in the appropriate reverse sequence.

User Request Handling

When RTSE is running, RM performs User Request Handling. RM handles the User login/logout request. For a successful login, the caller will receive a UserHandle number from RM. For the same user raising any other request, the request coming into RM should include that UserHandle for identification. In specific implementations, exchanges of the UserHandle will be encrypted to increase security. For every single user request, RM will check if the user can perform such a function-call and also check if the user can access (read or write) to the target media object (if applicable) based on such parameters as: the system's security level, the user's profile, and/or the target media object's properties. If the security control is passed, RM will also enforce some other

restriction specified in the software license key; for example, a user cannot access Mpeg2 media object if the key specified MP3 files only. RM handles most of these inter-manager tasks, eliminating most of the need for inter-manager direct communication.

If the checks are passed, in specific embodiments, RM will check if the request's parameters have to be transformed. For example, RM may convert file paths containing '~' to a user's home directory (e.g. /usr/username) using data stored in the user's profile to support a user home directory shortcut without requiring each file manager to be aware of a particular request's user 'home directory' or of the home directory shortcut notation.

Finally, if all checks are passed, then RM will pass the request and parameters to the appropriate manager(s) to handle the request. In specific embodiments and in some instances, a UserHandle will be removed from the parameters list before passing to the appropriate manager(s), while RM will keep track of the user information for returning results.

RM Handling of user requests

Most of the user request will be in the general forms (-> indicates data flow):

```
user request -> RM -> the appropriate manager;
results from the appropriate manager -> RM -> user.
```

However, for some complex request, the RM may have to call multiple managers or helpers, such as:

```
user request ->    RM -> the appropriate manager 1
                   RM -> the appropriate manager 2
                   RM -> the appropriate manager 3;
```

or

```
user request (e.g. XCOPY)  -> RM ->
   the appropriate master  -> the appropriate manager 1
                           -> the appropriate manager 2.
results from different manager -> RM (packaged the results) ->
   user.
```

For many requests, Streaming and File Managers need not to talk to other managers directly. However, in some instances, streaming manager will directly send requests to file manager (via File System Master in some embodiments) for real-time data retrieval and/or data storing. However, in general, other managers do not talk to

one another, but instead direct requests to RM, which translates and passes messages to other managers.

Regarding the "Streaming Manager -> File Manager" communication, it is originated by a user request through RM to the Streaming Manager to start streaming a media file. Therefore, RM has already ensured that the user request is under the security control and that's why this inter-manager communications need not to subsequently involve RM.

Event/Session Handling

When the RTSE is running, event/session message handling is performed by RM. RM can receive event/session messages from all other managers and in specific instances can consolidate and possibly redistribute messages. Here, messages is just one way of information passing, not a request. Therefore, the sender does not expect a result or reply.

If RM receives an Event message, it will send it to the appropriate Event Logging Module(s). Also, if a main executable (e.g. Server UIM) is registered by RM to receive an event message, RM will forward the message to that executable. Likewise, if RM receives a session message, it will send it to the appropriate Session Logging Module(s). Also, if the main executable (e.g. Server UIM) is registered by RM to receive session message, RM will forward the message to that executable. Therefore, various managers need not to talk to Event/Session Logging Modules directly.

STREAMING FILE MANAGER (FM)

Conventional file systems, such as Microsoft's FAT or NTFS, generally do not provide sufficient performance or access services for handling continuous media data. To address these issues, a Streaming File Manager according to specific embodiments of the present invention is used and provides an overhead-reduced file system kernel designed to perform real-time (e.g. time-critical) media streaming. The module is simple but powerful enough to handle real-time streaming requests.

However, commercial file systems such as Microsoft FAT or NTFS do provide some benefits. For example, many third party vendors develop applications and utilities on top of these file systems. To capture these benefits, while allowing for acceptable streaming performance, an FM according to specific embodiments of the present invention supports both time-sensitive Streaming File Systems (superior support for

continuous media streaming) and one or more widely-used conventional commercial file systems (one generalized example is a Window File System). A Streaming File System can support high-performance streaming service and the Window File System can provide less-powerful streaming but rich third party support for file manipulation.

According to further specific embodiments of the present invention, instead of simply supporting two individual file systems to meet different needs, the present invention may also create a proxy relation between these two file systems. In this embodiment, a Cache Module views the Window File System as a media archival and Streaming File System as a proxy streaming engine.

In this design, all media content manipulation (such as backup, file transfer) can be either by file system provided commands, file system provided utilities, or third party applications. In the meantime, the Streaming File System can act as a proxy to handle real-time streaming services.

From the teachings provided herein, it will be understood that the Streaming File Manager and the Encoding File Manager can generally be referred to as Streaming Source Managers. In specific embodiments, both types of managers are accessed through the same File System APIs and to external modules may be accessed as generic streaming data sources.

FM System Diagram and Components

FIG. 6 is a block diagram illustrating an example system diagram of a Streaming File Manager according to specific embodiments of the invention. This example is provided as a specific embodiment of a file manager according to the present invention, but it should be understood that other File Manager configurations can operate as part of the invention as shown in the previous figures.

The function of each module in this example FM according to specific embodiments can be understood as follows. It will be apparent from the teachings herein that not every module shown is required for each instance of an FM. An FM providing interface to a Windows File System may not need a Media Disk Handler that provides formatting, for example, because those functions are provided by the underlying file system.

Media Disk Handler can configure a storage device to be used as streaming storage for a streaming file system. This consists of two major tasks: format and

performance measurement. As mentioned above, RTSE needs to guarantee the streaming service, therefore, knowing the performance limitation on the underlying storage device is a pre-requisite. For a proprietary Streaming File System, the file format and storage allocation is totally under RTSE control and is not compatible with commercial file systems. To layout the disk as Streaming File System format, a configuration module provides a format function for formatting the storage in a Streaming File System Format. (As discussed below, for Window File Systems or other standard file systems, there is no need to perform a streaming specific "format" operation.)

API Handler interacts with Real-time Reception Manager to receive all file system service calls. It may call related internal modules to perform the service call or forward the service call to appropriate modules.

Real-time Scheduler: When there are many streaming sessions running concurrently, RTSE needs to arrange the service sequence among them to ensure every session can maintain the necessary quality of service. Real-time Scheduler is responsible for this. Because every session requires various amount of system resource (for instance, a session requesting a high quality MPEG-2 video consumes more resources than the one asking for low quality MP3 music clip), the module must ensure the resource is used in a consistent and coherent way.

Buffer Manager manages data buffers reserved for streaming purposes. In a particular embodiment, each streaming session has its own buffer to store the media data retrieved by Disk Scheduler.

Admission Control controls the license issue at this point in the RTSE. There are two levels of license control in RTSE. One level controls the whole system license. The other level controls the license issue for each individual media file. For the whole system license, RTSE uses two parameters to do the license protection: maximum number of streams and maximum streaming bandwidth. For each individual media file, RTSE limits the maximum number of streams accessing the file. Therefore, when a new request comes in and is passed to Admission Control for verifying. The module ensures the new request will satisfy both levels of license control.

Disk Scheduler arranges the IO requests and performs the storage read/write. To take advantage of multiple processors architecture on many machine platforms,

RTSE can generate multiple Disk Scheduler threads and have them work concurrently to improve overall system performance. Having multiple threads also facilitates balanced system operation.

Cache Manager manages cache arrangements, especially between the streaming file system and the conventional file system when that system is used as an archive for the streaming file system. Cache Manager receives requests for streaming media files and fetches profiles of the requested media file. In specific embodiments, Cache Manager in a streaming FS File Manager can determine whether a media file exists in the cache space (e.g. Streaming File System) and if not issue requests to a conventional File System to download the media file into a Streaming File System and after download has been started, returns the media profile information to API Handler. In a non-streaming file system, Cache Manager can receive the requests from the streaming FS manager and coordinate delivery of the data to the streaming FS.

The modules shown as Windows File System API and Streaming File System API are the APIs that handle the interface with various underlying file systems.

Example Operation Of FM

Case 1: Real-time Reception Manager issues a playback request for a particular media file.

1. API Handler receives the request and issues a request to Cache Manager to get the profile of the requested media file.
2. Cache Manager checks whether the media file exists in the cache space (e.g. Streaming File System).
3. If Cache Manager cannot find media file in the cache space, it issues a request to the conventional (e.g. Windows) File System to download the media file into Streaming File System. After download has been started for a period (e.g. a couple of seconds), the Cache Manager returns the media profile information to API Handler.
4. API Handler receives the media profile information and asks Admission Control module to verify the operation, e.g. it asks Admission Control whether a user can access the file at the present time without violating operating parameters.

5. Admission Control checks the license setting and make sure that admitting the new request will not violate the license setting for the whole system as well as for the requested media file.
6. If approved, API Handler passes the request to Real-time Scheduler to ask the scheduler to create a streaming session.
7. Real-time Scheduler sends a buffer allocation request to Buffer Manager.
8. If Buffer Manager approves the request, it returns a block of memory space, and the memory space pointer is returned.
9. Real-time Scheduler gets the memory space pointer and reserves the memory space for the newly created streaming session, creates a unique session ID for the request, and puts the session into the session schedule list for service.
10. If session created successfully, the unique session ID will return to the caller (Real-time Reception Manager).
11. If a session is created successfully, the Real-time Scheduler running in the background to continue its scan through the session scheduling list will interact with other modules to satisfy each scheduled session need.

Case 2: Playback the requested media file after the playback request is granted.

1. API Handler receives the read request for a session (granted in Case 1) from Real-time Reception Manager.
2. It passes the request to Real-time Scheduler asking for the required data.
3. Real-time Scheduler checks the session ID and determines where the data is stored for the particular read operation and updates its internal session structure to reflect the current media file offset location.
4. API Handler returns the data to the caller.

Case 3: Reception Manager issues a recording request for a particular media file.

1. API Handler receives the request and issues a request to Cache Manager to check the existence of the file in current file system.
2. If the file is not available, API Handler asks Admission Control module to verify the operation.

3. Admission Control checks the license setting and make sure that admitting the new request will not violate the license setting for the whole system.
4. If approved, API Handler passes the request to Real-time Scheduler to ask it to create a streaming session.
5. Real-time Scheduler sends buffer allocation request to Buffer Manager.
6. If Buffer Manager approves and returns a block of memory space, the memory space pointer is returned.
7. Real-time Scheduler gets the memory space pointer and reserves the memory space for the newly created streaming session. Then, it creates a unique session ID for the request and put the session into the session schedule list for service. In the meantime, it updates the system media profile to reflect the new media file information.
8. If the session is created successfully, the unique ID will return to the caller (Real-time Reception Manager).
9. If the session is created successfully, the Real-time Scheduler running in the background continues scanning through the session scheduling list and will interact with other modules to satisfy each scheduled session need.

Case 4: Download a file from Window File System into Streaming File System

1. Cache Manager issues a get media profile operation to Window File System for a particular file.
2. Cache Manager asks Admission Control module to verify the operation.
3. Admission Control checks the license setting and makes sure that admitting the new request will not violate the license setting for the whole system as well as for the requested media file.
4. If approved, Cache Manager passes the request to Real-time Scheduler to ask it to create a streaming session.
5. Real-time Scheduler sends buffer allocation request to Buffer Manager.
6. If Buffer Manager approves and returns a block of memory space, the memory space pointer is returned.
7. Real-time Scheduler gets the memory space pointer and reserves the memory space for the newly created streaming session. Then, it creates a unique session ID for the request and puts the session into the session schedule list for service.

8. If session is created successfully, the unique ID will return to the caller.
9. Cache Manager uses the session ID to continuously retrieve data by issue read operations to Real-time Scheduler.
10. After accumulating a period of data (such as a couple of second's worth of data), Cache Manager creates another recording request (specific to Streaming File System) similar to Case 3.
11. The newly created recording session retrieves the data from data accumulated by playback request (at Window File System).
12. This producer and consumer operation forms a data pipe between Window File System and Streaming File System.
13. The download operation completes when all data is in Streaming File System.
14. Streaming File System updates its internal structure to reflect the new media file.

In a further embodiment, as will be understood from the teachings herein, the FM interface shown in FIG. 6 is duplicated in separate instances for different conventional file systems and different streaming file systems. In specific embodiments, it will be understood that there is a separate FM for each file system and may also be a separate encoder instance for each encoder.

ENCODING MANAGER (EM)

Underneath the Real-Time Encoding API lies the implementation of real-time Encoding Manager/Module. FIG. 7 is a block diagram illustrating an example encoding manager according to specific embodiments of the invention. An example flow chart of Encoding Module operation is shown in FIG. 8. This module provides encoding device initialization/termination, logical mapping of encoding device to file system object, supporting file system semantics, routing control messages, set/get logical encoding parameters, and generating video/audio bit-stream. According to specific embodiments, there are two logical objects within the Encoding Module, e.g., Encoder Manager and Configuration Manager. A specific example of runtime procedures may be understood as follows.

1. **Scan installed encoder(s):**

The scanning process enables the system to recognize installed encoding devices, either hardware or software implementations. During the scanning process, the

Encoding Module calls an API "GetDeviceCount" defined in Encoder Manager for known encoder driver(s). The implementation of the API returns the number of encoding devices available. The Encoder Manger also assigns unique "Selector ID" and "Device ID" for each encoding device. The Selector ID is pre-registered in the Encoding Module for each driver, but the Device ID is determined at runtime. Thus, any addition or removal of any hardware or software encoding device will be automatically recognized at this discovery process. It also provides a logical enumeration of encoding devices, free from vendor-specific or system-specific device enumeration.

2. **Instantiate encoder(s):**

Once the system recognizes the existence of encoding devices, program control or user interaction is used to create encoder instance(s). The Encoder Manager manages every instance. Thus, all of the control operations (enable, disable, start, stop, pause, and resume) are transparent to the callers for the heterogeneous encoding devices.

3. **Set/Get logical encoding parameters:**

Each encoding device has its own capability and format in encoding video/audio bit-stream. However, there are set standards by ITU, ISO, or vendors. Though the values of the settings vary by encoders, the general physical requirements do not change. For instance, the input connector types are standard ones. The color adjustments for video are well-known coloring schemes. The audio adjustments are also adhered to audio engineering standards. Hence, Configuration Manager defines and manages a logical parameter set. The translation of these parameters to device-specific settings is also provided by Configuration Manager so that it is transparent to the callers, and can be tagged by the Selector ID.

4. **Apply parameter changes:**

When the parameter change is going to be committed to the encoding device, Encoder Manager works in accord with Configuration Manager. Configuration Manager translates the settings to physically applicable values for the target device. Encoder Manager applies the translated values to the target device. This logical functionality separation between Encoder Manager and Configuration Manager ensures that the Encoding Module can be further extended or scaled down. In a

scaled down application, the application might have several sets of pre-defined non-changeable parameters that Encoder Manager at runtime can use directly. Such application may run on embedded system or memory-stringent environments.

5. **Generate encoded bit-stream(s):**

Bit-stream generation is one of the most important end results. However, due to hardware or software implementation in encoding device, handling of the bit-stream varies. Some implementations require installation of software callback functions to retrieve hardware-generated data in real-time; other implementations use an asynchronous software event object to signal the availability of real-time encoded data. In order for the applications running on top of the Encoding Module to be able to pick up data without worrying about timing issues even on a non-real-time operating system, Encoder Manager manages the per-encoding-instance circular memory buffer for interrupt-driven or event-driven bit-stream handling. Data available through both handling mechanisms are being redirected to the memory buffer.

6. **Data redirection for in-memory circular buffer:**

To ensure a consistent and coherent access to the bit-stream, the Encoder Manager provides a transparent API "GetEncoderBitStream" to access in-memory circular buffer. The implementation uses a "laps" concept so that the callers know the bit-stream offset from the beginning of encoded bit-stream. Hence, it provides a synchronization mechanism for callers regardless of the timing or data. It also adheres to the general semantics used in file system internal. Data can be copied or redirected to the other streaming system modules or written to the file systems that are supported by the operating system.

RTSE FURTHER SAMPLE SEQUENCES

The above managers collaborate with each other to achieve large-scale streaming service. The engine is powerful enough to handle various types of real-time file systems, various types of software and hardware encoding modules, various types of database engines, various types of networking protocols, various types of real-time media, and various types of streaming engines. The example engine illustrated in FIG.

4 and FIG. 5 are for illustration purposes. The example sequences described below are further for illustration purposes.

Sample Sequences of Client/Server Interactions

Case 1: Client requests to play back pre-recorded real-time media from server

1. Client requests to play back Movie A via web browser or other connection.
2. The request is transmitted to RTSE's Network Manager via Internet or Intranet or other communication channel.
3. Network Manager receives the request and verifies the security level of the remote client and the client identity.
4. If the client has enough access authority, the request is passed to Streaming Manager via Reception Manager.
5. Streaming Manager records down the active stream and passes the request to Streaming File Manager.
6. Streaming File Manager verifies that there are sufficient resources available to handle delivery of the requested stream and decides to accept or reject the current request.
7. If accepted, Streaming File Manager starts the real-time scheduling to retrieve the requested media to Streaming Manager.
8. Streaming Manager starts the real-time scheduling to deliver requested media to remote client via pre-negotiated network protocols.
9. Client receives the data and presents the pre-recorded media to a user.
10. According to specific embodiments of the present invention, a client may allow a user to perform virtual VCR functions such as *index seek, fast forward, backward, time seeking, pause*, etc. These requests reach Network Manager, which passes them through Reception Manager to Streaming Manager to control the stream.

Case 2: Client requests to play back live real-time channel from server.

1. Client requests to play Channel B via web browser or other connection.
2. The request is transmitted to RTSE's Network Manager via Internet or Intranet or other communication channel.
3. Network Manager receives the request and verifies the security level of the remote client and the client identity.

4. If the client has enough access authority, the request is passed to Streaming Manager via Reception Manager.
5. Streaming Manager records down the active stream and passes the request to an Encoding File Manager.
6. Encoding File Manager verifies that there are sufficient resources available to handle delivery of the requested stream to decide to accept or reject the current request.
7. If accepted, Encoding File Manager starts the real-time scheduling to deliver the requested media to Streaming Manager.
8. Streaming Manager starts the real-time scheduling to deliver requested media to remote client via pre-negotiated network protocols.
9. Client receives the data and presents the pre-recorded media to a user.
10. According to specific embodiments of the present invention, a client may allow a user to perform channel surfing functions or other functions appropriate for live data. These requests reach Network Manager, which passes them through Reception Manager to Streaming Manager to control the stream.

Case 3: Local Management Console communicates with RTSE

1. Local Management Console will first check if RTSE is running.
2. If it is not running, the Console will bring it up, and the RTSE Reception Manager will start to listen for further commands.
3. Users of the console have to identify themselves through the console's login function.
4. After login, RTSE is ready to report its latest status to the console.
5. In specific embodiments, the operator may start RTSE services fully or partially using the console, depending on operator needs.
6. The console provides Client Connection Management, Media Object (files or live channels) Management, Server Settings & Managers Configurations.
7. The console may also facilitate Event Log Viewing and Performance Monitoring.
8. According to specific embodiments of the present invention, if the operator only wishes to monitor RTSE operation, the operator may login as a monitoring-only user and allow monitoring windows to run, without risking any unauthorized access while the operator is away from the console.

9. When quitting the console, the operator can choose whether to stop RTSE or not.
10. If RTSE is stopped, all the client connections will be terminated.

Case 4: Remote Management Console communicates with RTSE

1. Remote Management Console issues an HTTP-based (or other protocol, such as SNMP) "check server status" request to RTSE's HTTP (or other) Listener.
2. If RTSE is not running, the HTTP request will be timed out and the Remote Management Console reports the result.
3. If RTSE is running, its HTTP Listener will interpret the request and identify the user's rights through a login function.
4. After login, HTTP Listener issues request of "check server status" to Real-time Reception Manager to get the server status streaming status back to HTTP Listener.
5. HTTP Listener then packs the result into an HTML page back to the Remote Management Console.
6. Remote Management Console displays the result to Console's display.
7. The Remote Management Console can periodically query RTSE status.
8. In addition to "check server status" request, Remote Management Console has requests such as "get streaming performance", "get individual or overall streaming sessions information", "get streaming events", "remotely import media file into RTSE", "remotely export media file out of RTSE", etc.

Case 5: Proxy Example

1. Client requests to play back movie A via web browser or other connection.
2. The request is transmitted to RTSE's Network Manager via Internet or Intranet or other communication channel.
3. Network Manager receives the request and verifies the security level of the remote client and the client identity.
4. If the client has enough access authority, the request is passed to Streaming Manager via Reception Manager.
5. Streaming Manager records down the active stream and passes the request to Streaming File Manager.
6. Streaming File Manager verifies that the requested media is available on a Windows or other non-streaming conventional File System.

7. If the request media is not available on a conventional File System, the request is denied.
8. Determine the streaming resource for two different cases: (a) the requested media is in conventional File System and has been cached in Streaming File System; or (b) the requested media is in conventional File System but has not been cached in Streaming File System.
9. Streaming File Manager verifies the storage resource to decide to accept or reject the current request.
10. If approved, for case (a) in item 8, Streaming File Manager starts the real-time scheduling to retrieve the requested pre-recorded media to Streaming Manager.
11. If approved, for case (b) in item 8, Streaming File Manager starts the real-time scheduling to download the requested media from conventional File System into Streaming File System. After a portion of data has been cached in the Streaming File System, Streaming File Manager starts the real-time scheduling to retrieve the requested media (just cached) to Streaming Manager.
12. Streaming Manager starts the real-time scheduling to deliver requested pre-recorded media to remote client via pre-negotiated network protocols.
13. Client receives the data and starts play back of the pre-recorded media
14. According to specific embodiments of the present invention, a client may allow a user to perform virtual VCR functions such as *index seek, fast forward, backward, time seeking, pause*, etc. These requests reach Network Manager, which passes them through Reception Manager to Streaming Manager to control the stream.

Case 6: Database Example

1. Access Control module in Reception Manager requests to find a specific user profile by a user-name.
2. Data Management first locates the location and access mechanism of the User Profile Table from server settings and configurations.
3. If the access mechanism is ODBC, DBM will bring up UserDbODBC to follow up the request, UserDbODBC may get the Profile from all types of DBMS servers which support ODBC.

4. If the access mechanism is DAO, DBM will bring up UserDbDAO to follow up the request. The UserDbDAO is designed to use DAO to talk to Microsoft Jet Database in an optimized way.
5. If the access mechanism is LDAP, DBM will bring up UserDbLDAP to follow up the request, UserDbLDAP will interface to the corresponding LDAP server to access a distributed data repository.
6. Data Manager will return result to the Access Control in the same manner, no matter what access mechanism has been used and where the data stored.

Example RTSE (Server) StartUp Procedure

According to specific embodiments of the present invention, as an example, the following steps are performed when an example RTSE server (which may be understood as generally configured as shown in FIG. 4 or FIG. 5) starts and then a client connects:

1. **Reception Manager Initialization**

Internal modules inside the Reception Manager are initialized and passed back their stored settings from non-volatile storage to restore their previous states. Messaging and logging features are then initialized to handle and store any messages coming from other managers. User Authentication and Access Control are then turned on to guard and protect the incoming requests from the managers using the Server Reception API. The Server Master, Streaming System Master and File System Master are invoked and ready to communicate with Streaming Managers and File/Encoding Managers. After the initialization, all managers are ready to report their current status to Reception Manager for central control.

2. **Operator Login**

Before any further Server Reception API function will be served, a user needs to login with an operator role. User Authentication module maintains user profiles / user groups and stores their operation rights. In specific embodiments, this may be accomplished through use of external database store using the DAPI. Access control module will authorize a request from the logged-in user by the request operation type and the access rights about the involved media objects if any. Unauthorized requests will be rejected immediately.

3. **Server Configuration**

If necessary, Reception Manager & other mangers are ready to be configured to the user's preferences. General configurable setting includes aspect of Access Control, Networking, Media Profile, and Logging. Individual manager specific configuration may also be done if needed, e.g. video volume formatting, etc.

4. **Server Initialization/Start**

Different combination of streaming/file/encoding/network managers may result in different functionality of the server for different application requirements. Each manager has initialization and startup procedure to enable their services. Reception Manager allows these services to be started in a very flexible way.

5. **Server Performance Monitor**

Managers report their performance details when started to RM. Reception Manager redirects this data flow to any parties needed.

6. **Media Object Management**

Through Reception Manager with access rights checking, Media Objects among file mangers may be freely copied, pasted, deleted, moved, created, imported and exported by authorized users. Media Objects in all file mangers supports same entry point and hierarchical structure. Each Media Object Profile contains such parameters as media format details, updateable clip title, author, descriptions, license and its user/group access rights. Media Access Control is also applied to directory levels.

7. **Client Connect**

While a networking manager is up and running, management or playback clients can establish connections to the server machine and talk to the Reception Manager directly. Each connection can be logged and monitored clearly through Reception Manager.

8. **Client Login**

The Reception Manager requires each connection to have a valid login. Further requests from the connection will be controlled by a logged-in user's profile. Each login session's details will be stored via the Access Control module.

9. **Client Media Object Browsing**

Clients may browse into all File Mangers' media object directories through the Reception Manager. RM will filter out those files that the user has no authority to browse.

10. **Client: Start Media Object Streaming**

The Reception Manager accepts client's command to start streaming through a specific streaming manger for a specific file manager's media object. The two managers will coordinate with each other to perform the streaming smoothly. Connection's status will be changed and monitored as needed.

11. **Client Quit: Streaming Stop, Logout, and Disconnect**

Reception Manager sends messages to other managers to clean the related connection. Login control also reflects the login session removal.

12. **Reception Manager Quit: Stop Server, Uninitializing Server, and Release Resource**

Reception Manager will stop the managers in appropriate sequence, then clean up its internal modules.

EMBODIMENT IN A PROGRAMMED INFORMATION APPLIANCE

The invention can be implemented in hardware and/or software. In some embodiments of the invention, different aspects of the invention can be implemented in either client-side logic or a server-side logic, though the present invention will usually be implemented in a server side device. As will be understood in the art, the invention or components thereof may be embodied in a fixed media program component containing logic instructions and/or data that when loaded into an appropriately configured computing device cause that device to perform according to the invention. As will be understood in the art, a fixed media program may be delivered to a user on a fixed media for loading in a user's computer or a fixed media program can reside on a remote server that a user accesses through a communication medium in order to download a program component.

FIG. 8 shows an information appliance (or digital device) 700 that may be understood as a logical apparatus that can read instructions from media 717 and/or network port 719. Apparatus 700 can thereafter use those instructions to direct server or client logic, as understood in the art, to embody aspects of the invention. One type of

logical apparatus that may embody the invention is a computer system as illustrated in 700, containing CPU 707, optional input devices 709 and 711, disk drives 715 and optional monitor 705. Fixed media 717 may be used to program such a system and may represent a disk-type optical or magnetic media, magnetic tape, solid state memory, etc. The invention may be embodied in whole or in part as software recorded on this fixed media. Communication port 719 may also be used to initially receive instructions that are used to program such a system and may represent any type of communication interface, including telephone modem, network interface, ADSL or cable modem, or wireless interface.

The invention also may be embodied in whole or in part within the circuitry of an application specific integrated circuit (ASIC) or a programmable logic device (PLD). In such a case, the invention may be embodied in a computer understandable descriptor language which may be used to create an ASIC or PLD that operates as herein described.

Thus, it will be understood from the teachings provided herein, unlike commonly used streaming servers, a streaming system constructed according to specific embodiments of the present invention, can handle generic streaming file systems by adapting to those systems through a File Manager and can also utilize non-streaming, or conventional file systems.

Similarly, a streaming system according to the invention, can handle streaming data from generic hardware and software encoding modules. The architecture generally described in FIG. 3 and more particularly in FIG. 4 and FIG. 5 also allows a streaming system to handle generic database engines for processing real-time media and various networking protocols to perform streaming service. Likewise, a system according to specific embodiments of the present invention, can schedule large-scale real-time time media concurrently from mass storage, to local memory and remote clients via Internet, Intranet, Cable, and Wireless connections.

As described herein, according to specific embodiments, the present invention also provides for a Real-Time Encoding API wherein the module that implements the API, provides unique software plug-and-play features to scan and instantiate underlying multi-vendor encoding devices. The encoding device can be hardware-based or software-based encoder and may be independent of any system bus architecture.

The invention provides a coherent system design to address different market segments. Services can be developed based on a generic File System API, and that File System API can be adapted to work with any real-time bit-stream generating device. Thus, a server according to the invention can enable applications such as Home Security, Secured Live Broadcasting (for financial institutions, government agencies, etc.), and In-house Broadcasting (for mid to large-size corporations and educational institutions), and Remote Sensing & Monitoring.

A streaming system according to the invention also easily allows extension to new technologies, such as additions for new encoding hardware/software, new video formats, new file systems, new streaming protocols, etc. Such new technologies can work seamlessly through the File System API without compromising or rewriting the other collaborating modules/managers.

According to further specific embodiments, the present invention provides a basis for a Software Development Kit (SDK) for hardware and software vendors that allows vendors to program their hardware or software independent of the rest of the real-time streaming system. New hardware or software encoding solution can be introduced and integrated with existing or new applications/utilities simply by adapting the new solutions through the File System APIs.

OTHER EMBODIMENTS

The processes, sequences or steps and features discussed above are related to each other and each are believed independently novel in the art. The disclosed processes and sequences may be performed alone or in any combination to provide a novel and unobvious system or a portion of a system. It should be understood that the processes and sequences in combination yield an equally independently novel combination as well, even if combined in their broadest sense; i.e. with less than the specific manner in which each of the processes or sequences has been reduced to practice as described herein.

The streaming media delivery server module as described herein, in accordance with one aspect of the present invention is robust, operationally efficient and cost-effective. In addition, the present invention may be used in connection with presentations of any type, including sales presentations and product/service promotion, which provides the video service providers additional revenue resources.

While the forgoing and attached are illustrative of various aspects/embodiments of the present invention, the disclosure of specific sequence/steps and the inclusion of specifics with regard to broader methods and systems are not intended to limit the scope of the invention which finds itself in the various permutations of the features disclosed and described herein as conveyed to one of skill in the art.

WHAT IS CLAIMED:

1. A method of providing streaming services from a flexible and expandable streaming server architecture comprising:

using a central reception manager to communicate with a plurality of cooperating semi-autonomous interface managers, wherein said interface managers handle different types of operations relating to streaming services;

using a file systems interface to handle communication of data with one or more data sources and to handle exchange of said data with a streaming interface manager; and

using said streaming interface manager to exchange time-critical streaming data with a communications network.

2. The method of claim 1 wherein said central reception manager receives user requests and distributes those requests to various appropriate other managers.

3. The method of claim 1 wherein said one or more data sources comprise:

at least two different file systems, including at least one streaming file system.

4. The method of claim 1 wherein said one or more data sources comprise:

at least two different file systems, including at least one streaming file system and at least one conventional file system.

5. The method of claim 1 further comprising:

using a network interface to communicate with one or more clients via a network; and

using user and database interfaces along with user authentication and access control logic to authenticate a user and a media request.

6. The method of claim 1 further comprising:

using said file systems interface to communicate with a file system manager for a conventional file system and a file system manager for a streaming file system, said file system managers each providing admission control, cache management, buffer management and scheduling.

7. The method of claim 1 wherein said file systems interface provides a standard interface to said central manager and said streaming interface managers while allowing integration of various streaming and conventional file systems.

8. The method of claim 1 wherein said file systems interface provides a standard interface to said central manager and said streaming interface managers while allowing integration of various live data streaming encoders.

9. The method of claim 1 wherein there are multiple instances of said file systems interface, each instance providing an interface for a different file system.

10. A system providing streaming data services comprising:

a central manager;

a plurality of other managers providing interfaces to various services;

wherein at least one of said services comprises exchange of streaming data; and

wherein said central reception manager provides application programming interfaces to said plurality of other managers.

11. The system of claim 10 wherein said other managers comprise:

a streaming manager providing communications of one or more streaming sessions according to one or more protocols; and

a file manager providing a standard interface to one or more data sources of streaming data.

12. The system of claim 11 wherein said streaming manager handles requests from said central reception manager and calls streaming file manager and an encoding file manager and further wherein said streaming manager takes the data from either streaming file manager or encoding file manager and schedules said data for delivery to remote clients via any pre-negotiated networking protocols.

13. The system of claim 11 wherein said file manager comprises:

an encoding file manager providing interface to one or more encoders.

14. The system of claim 11 wherein said file manager comprises:

an encoding file manager and encoding instances providing interface to one or more encoders.

15. The system of claim 13 wherein said encoders comprise generic hardware and software encoding modules.

16. The system of claim 13 wherein said encoding file manager bridges different hardware and/or software encoders to handle delivery of real-time data to said central manager via a generic file system API.

17. The system of claim 11 wherein said file manager comprises:
one or more streaming file managers providing interface to one or more file systems.

18. The system of claim 17 wherein said file manager further provides an interface for a streaming file system and a conventional file system.

19. The system of claim 11 wherein said other managers further comprise:
a network manager providing management and interface to one or more network listeners according to one or more network communication protocols.

20. The system of claim 11 wherein said other managers further comprise:
an administration manager providing interface to one or more administrative console functions.

21. The system of claim 11 wherein said other managers further comprise:
a user interaction manager providing interface to one or more user interaction console functions.

22. The system of claim 11 wherein said other managers further comprise:
a data manager providing interface to one or more data stores.

23. The system of claim 11 wherein said streaming manager provides interfacing to one or more streaming services provided by one or more protocols consisting of the group RTP Streaming, HTTP Streaming, and Real-time Streaming.

24. The system of claim 11 wherein said one or more data sources comprise two or more different streaming file systems.

25. The system of claim 10 wherein said central manager defines a set of communication protocols for said plurality of other collaborating managers to complete streaming and management requests from clients and consoles.

26. The system of claim 10 wherein said central manager performs the following:
retrieving stored data from a streaming data source manager;
processing requests from a network manager;
processing requests from an administration manager;
processing database requests to/from a database manager;
processing user interaction requests from a user interaction manager; and
processing requests from a streaming manager.

27. The system of claim 26 wherein said streaming data source manager comprises an encoding file manager that transfers live streaming data.

28. The system of claim 26 wherein said streaming data source manager comprises a streaming file manager that handles pre-stored streaming data.

29. The system of claim 11 wherein said streaming data source manager handles delivery of pre-stored data in a variety of storage systems and schedules concurrent stream requests from said streaming manager.

30. The system of claim 11 wherein said streaming file manager comprises:
an admission control module;
a scheduler;
a media disk module; and
a cache module.

31. A real-time reception manager comprising:
a streaming systems API providing interface to one or more streaming managers;
a server reception API providing interface to one or more other managers;
a data management API providing interface to one or more data sources; and

a file systems API providing interface to one or more streaming data source managers; and

a server master communicating with a streaming system master and a file system master.

32. The system of claim 31 further comprising:

a user authentication and access control module communicating with said database management API.

33. The system of claim 31 wherein said streaming managers further comprise:

one or more network managers;

one or more administration managers; and

one or more user interface managers.

34. A real-time encoding manager comprising:

an interface with a file system API;

one or more vendor hardware driver APIs;

one or more vendor software driver APIs; and

each of said hardware and software driver APIs communicating via an operating system to one or more drivers associated with one or more encoders.

35. A file manager for use in a streaming engine comprising:

a media disk handler that configures a storage device to be used for streaming storage, including formatting and performance measurement;

an interface handler for handling communications with outside modules;

a real-time scheduler that arranges service sequence among running streaming sessions to ensure that said sessions can maintain quality of service;

a buffer manager that manages data buffers reserved for streaming purposes; and

a disk scheduler that arranges I/O requests and performs storage read/write to a physical storage device.

36. The system of claim 35 wherein multiple disk scheduler threads may be generated and work concurrently to utilize a multiple processor architecture.

37. The system of claim 35 further comprising an admission control module that controls access to streaming data based on access policies.

38. The system of claim 35 wherein for a streaming file system, file format and storage allocation are under control of said file manager.

39. The system of claim 35 further comprising:

a cache module that views a non-streaming file system as a media archive and a streaming file system as a proxy streaming engine and wherein said streaming file system can act as a proxy to handle time critical streaming of files downloaded from said non-streaming file system.

40. A method of managing a real-time media service request at a streaming file manager comprising:

receiving a request at an interface handler of said streaming file manager;

issuing a request to a cache manager to retrieve a profile of a requested media file;

said cache manager determining whether said media file exists in a streaming file system;

if said media file is not found in said streaming file system, said cache manager issuing a request to a conventional file system to download said media file into said streaming file system and after download has proceeded for an interval, said cache manager returning said profile to said interface handler; and

passing said request to a scheduler to create a streaming session.

41. The method of claim 40 further comprising:

sending a buffer allocation request to a buffer manager;

if said buffer manager approves and returns a block of memory space, returning a memory space pointer;

said scheduler receiving said memory space pointer and reserving memory space for a newly created streaming session;

said scheduler creating a unique session id for said request and placing said session into a session schedule list for service; and

if said session is created successfully, returning said unique session id to a calling reception manager.

42. A method of managing a real-time media service request at a streaming file manager comprising:

receiving a request at an API handler of said streaming file manager to record a streaming file;

issuing a request to a cache manager to retrieve a profile of a requested media file;

accepting a recording session for said streaming file session.

43. A method of moving a file from a conventional file system to a streaming file system comprising:

issuing a get media profile operation from a cache manager to a conventional file system for a particular file;

issuing a request to a scheduler to create a streaming session;

if said session is created successfully, returning a session ID to said cache manager; and

said cache manager using said session ID to continuously retrieve data by issuing read operations to said scheduler.

44. The method of claim 43 further comprising:

after retrieval has proceeded for an interval, said cache manager issuing another request to said scheduler.

45. The method of claim 43 further comprising:

said scheduler sending a buffer allocation request to a buffer manager;

said buffer manager approving said request and returning a block of memory space pointed to by a memory space pointer;

said scheduler receiving said pointer and reserving memory space for a newly created streaming session;

said scheduler creating a unique session ID for the request; and

said scheduler placing said session into a session schedule list for service.

46. The method of claim 43 wherein a newly created recording session retrieves the data from data accumulated by playback request at said conventional file system.

47. The method of claim 43 wherein said request forms a data pipe between said conventional file system and streaming file system.

48. A method of handling a request for playing back streaming media comprising:

receiving a request to play back a media selection at a network manager via a network;

said network manager receiving said request and verifying a security level of a remote client identity;

passing said request to a streaming manager via a central reception manager interface;

said streaming manager recording the active stream and passing said request to a streaming source manager;

said streaming source manager verifying a storage resource to decide to accept or reject said request;

if approved, said streaming source manager starting real-time scheduling to retrieve requested media to said streaming manager; and

streaming manager beginning real-time scheduling to deliver requested media to remote client via negotiated network protocols.

49. The method of claim 48 further comprising;

receiving a request to perform a virtual VCR function at said network manager to control the status of the stream.

50. The method of claim 48 wherein said streaming source manager comprises a streaming file manager and said media selection comprises a prerecorded media file.

51. The method of claim 48 wherein said streaming source manager comprises an encoding file manager and said media selection comprises live real-time media.

52. The method of claim 50 wherein said streaming source manager verifying a storage resource further comprises:

determining if the requested media is available on a conventional file system and if not denying the request;

determining the streaming resource for two different cases: (a) the request media is in said conventional file system and been cached in said streaming file system; or (b) the requested media is in said conventional file system but not been cached in streaming file system;

if approved, for case (a) said streaming file manager initiating real-time scheduling to retrieve requested pre-recorded media to said streaming manager;

if approved, for case (b), streaming file manager initiating real-time scheduling to download the requested media from said conventional file system into streaming file system; and

after a portion of data has been cached in the streaming file system, streaming file manager initiates real-time scheduling to retrieve just cached requested media to streaming manager.

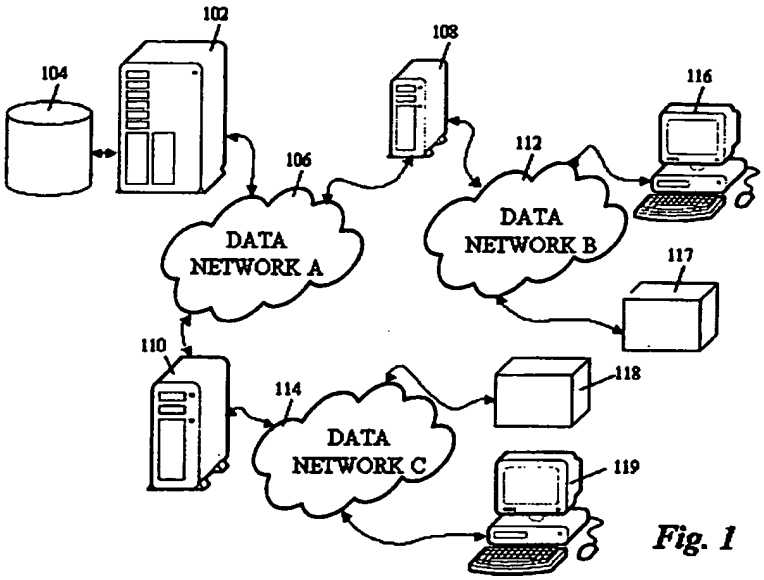


Fig. 1

FIG 1

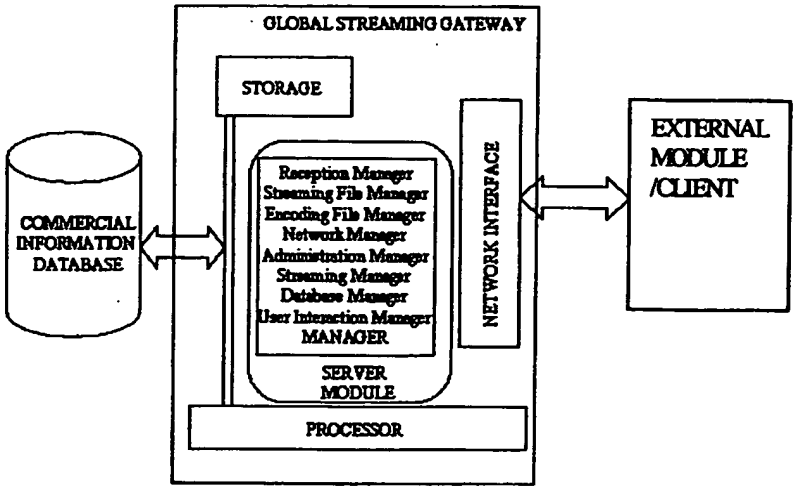
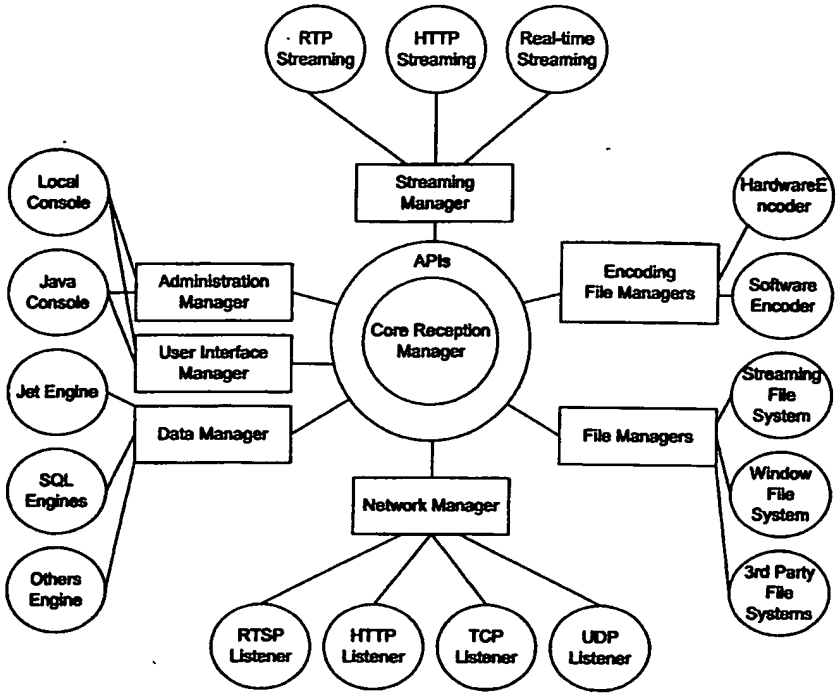


FIG 2



The Spider-Web Architecture of RTSE

FIG 3

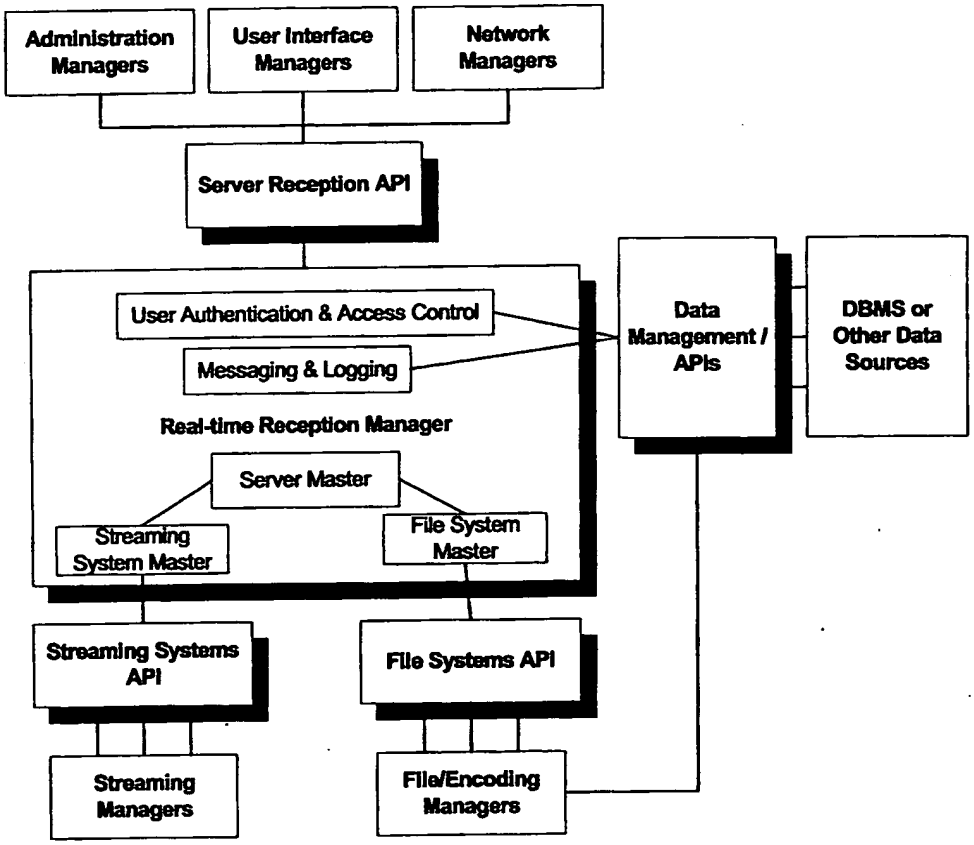

Administration Managers
User Interface Managers
Network Managers
Server Reception API
User Authentication & Access Control
Messaging & Logging
Real-time Reception Manager
Server Master
Streaming System Master
File System Master
Data Management / APIs
DBMS or Other Data Sources
Streaming Systems API
File Systems API
Streaming Managers
File/Encoding Managers

FIG 4

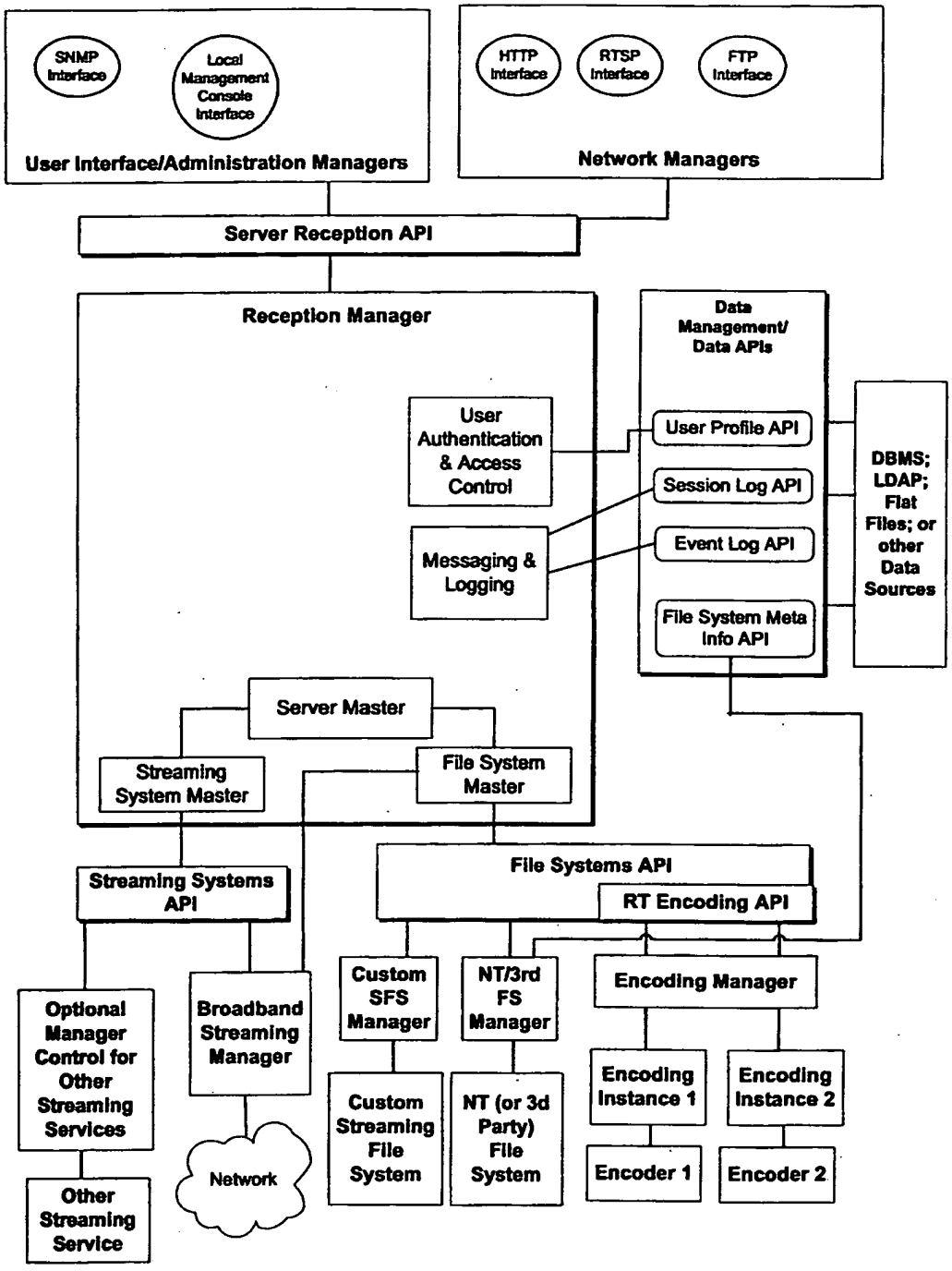

SNMP Interface
Local Management Console Interface
User Interface/Administration Managers
HTTP Interface
RTSP Interface
FTP Interface
Network Managers
Server Reception API
Reception Manager
Data Management/ Data APIs
User Authentication & Access Control
Messaging & Logging
User Profile API
Session Log API
Event Log API
File System Meta Info API
DBMS; LDAP; Flat Files; or other Data Sources
Server Master
Streaming System Master
File System Master
Streaming Systems API
File Systems API
RT Encoding API
Optional Manager Control for Other Streaming Services
Broadband Streaming Manager
Custom SFS Manager
NT/3rd FS Manager
Encoding Manager
Encoding Instance 1
Encoding Instance 2
Encoder 1
Encoder 2
Custom Streaming File System
NT (or 3d Party) File System
Network
Other Streaming Service

FIG 5

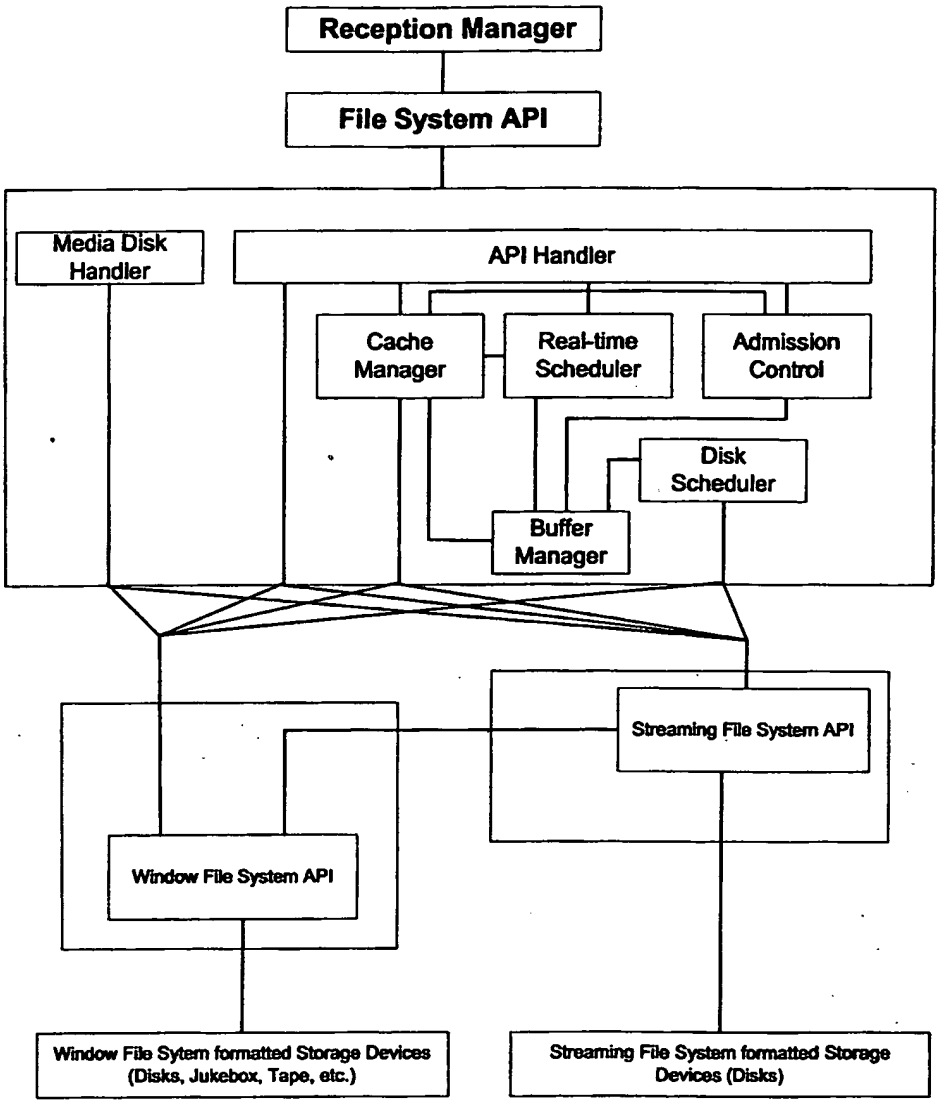

Reception Manager
File System API
Media Disk Handler
API Handler
Cache Manager
Real-time Scheduler
Admission Control
Disk Scheduler
Buffer Manager
Streaming File System API
Window File System API
Window File Sytem formatted Storage Devices (Disks, Jukebox, Tape, etc.)
Streaming File System formatted Storage Devices (Disks)

FIG 6

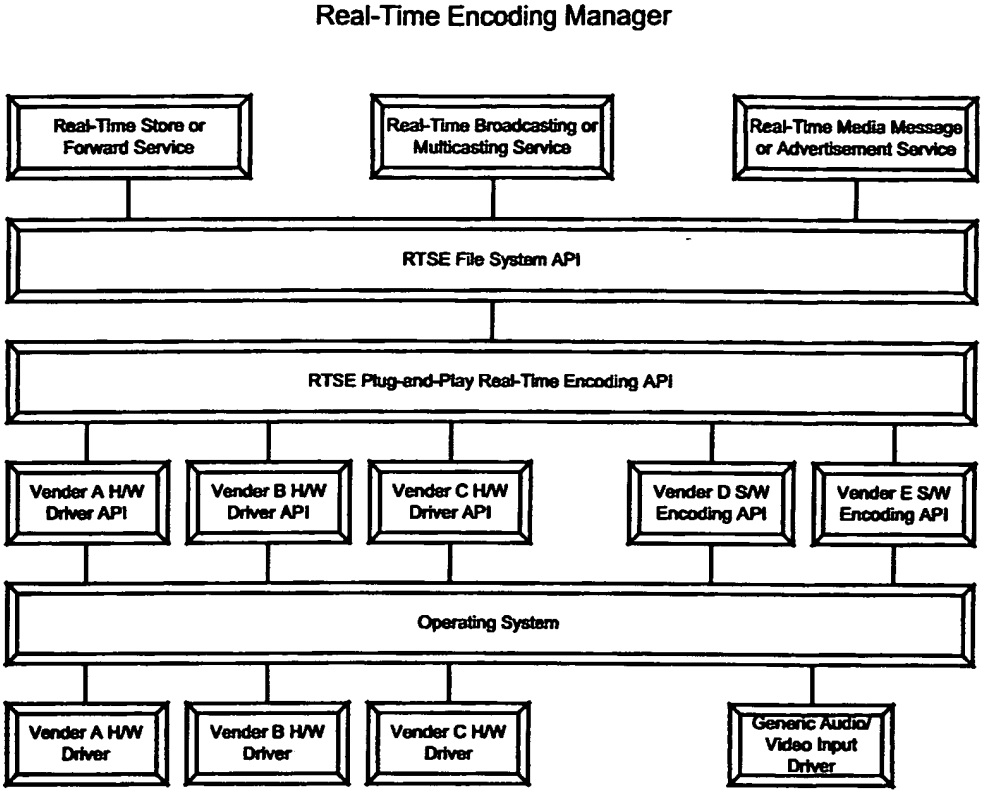

Real-Time Encoding Manager
Real-Time Store or Forward Service
Real-Time Broadcasting or Multicasting Service
Real-Time Media Message or Advertisement Service
RTSE File System API
RTSE Plug-and-Play Real-Time Encoding API
Vender A H/W Driver API
Vender B H/W Driver API
Vender C H/W Driver API
Vender D S/W Encoding API
Vender E S/W Encoding API
Operating System
Vender A H/W Driver
Vender B H/W Driver
Vender C H/W Driver
Generic Audio/ Video Input Driver

FIG 7

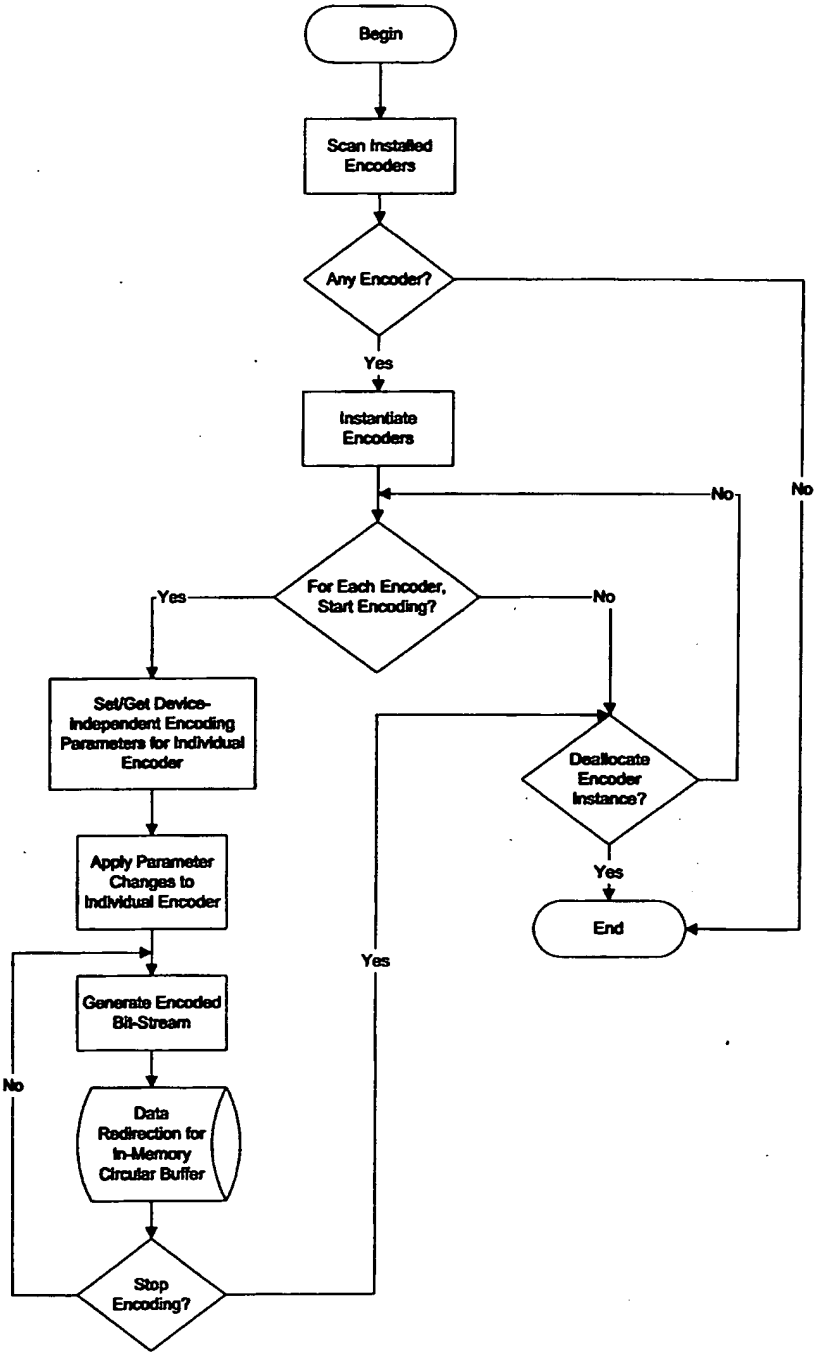

Begin
Scan Installed Encoders
Any Encoder?
Yes
No
Instantiate Encoders
No
For Each Encoder, Start Encoding?
Yes
No
Set/Get Device-Independent Encoding Parameters for Individual Encoder
Deallocate Encoder Instance?
Yes
No
Apply Parameter Changes to Individual Encoder
End
Yes
Generate Encoded Bit-Stream
No
Data Redirection for In-Memory Circular Buffer
Stop Encoding?

FIG 8

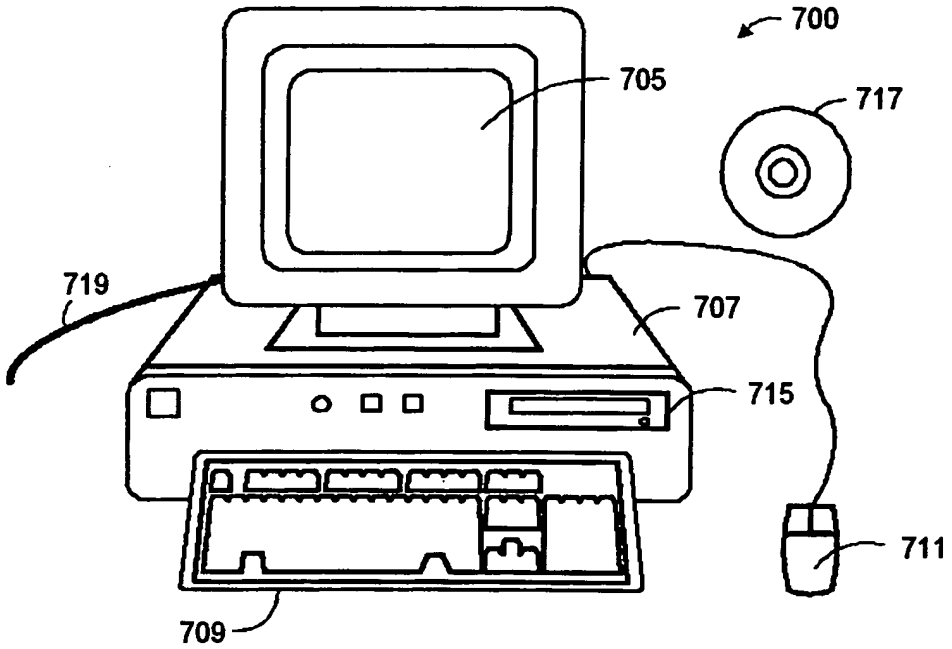


FIG 9